Andreas Havenstein
(andreas.havenstein@it-agile.de)
ist bei der it-agile GmbH Trainer, Senior-Softwareentwickler und Architekt. Seine Schwerpunkte sind flexible Architekturen und die Anwendungsentwicklung unter Einsatz agiler Methoden, insbesondere von XP.

Marko Schulz
(marko.schulz@it-agile.de)
unterstützt Teams dabei, agile Softwareentwicklungspraktiken umzusetzen. Neben Schulungen legt er bevorzugt selber mit Hand an und programmiert mit. Insbesondere beachtet er technische Grundlagen wie TDD und evolutionäres Design.

Certified Scrum Developer: Der ganzheitliche Entwickler

Scrum trat einst als allgemeines Projekt-Rahmenwerk an, mit dem sich nicht nur Softwareentwicklung betreiben lässt, sondern beispielsweise auch Marketing und Projektmanagement. In der Praxis merkten viele Teams jedoch, dass es schwer ist, mit Scrum zu arbeiten, wenn man nicht auch moderne Programmierpraktiken benutzt. Diesem Umstand zollte die Scrum-Alliance Tribut und führte den „Certified Scrum Developer" ein. So finden Teams noch besser zu einem runden Vorgehen nach Scrum. In diesem Artikel zeigen wir, welche Praktiken die Scrum-Alliance für Softwareentwickler empfiehlt, welche Gefahren drohen, falls sie sie nicht beherrschen, und welchen Nutzen sie daraus ziehen, wenn sie sie erlernen und einsetzen.

Als Mitte der Neunziger Jahre schwergewichtige, dokumentenlastige Prozesse den Markt dominierten, gab es eine Gruppe von Softwareentwicklern und Projektmanagern, die der Trägheit der Wasserfall-Prozesse eine leichtgewichtige Alternative entgegenstellen wollten. Kent Beck beispielsweise erkannte, dass die bis zum Maximum gesteigerte Anwendung bewährter Engineering-Praktiken in Kombination mit vielen kurzen Feedback-Schleifen ein Softwareprojekt zum Erfolg führen kann.

Er fasste Entwicklungspraktiken zum *eXtreme Programming* (XP) zusammen und schuf und beschrieb in [Bec00] den ersten großen Vertreter dessen, was später als agile Softwareentwicklung bezeichnet wurde.

Scrum entstand zu einer ähnlichen Zeit wie XP und wurde Anfang des neuen Jahrtausends bekannter, als Ken Schwaber ein erstes Buch hierzu veröffentlichte (vgl. [Sch01]). In den darauf folgenden Jahren setzte ein so großer Scrum-Boom ein, dass agile Softwareentwicklung und Scrum für viele synonym wurden. Entwicklungspraktiken, in XP noch dominierender Bestandteil, rückten in den Hintergrund. Scrum ist ein Methodenrahmen, der prinzipiell unabhängig

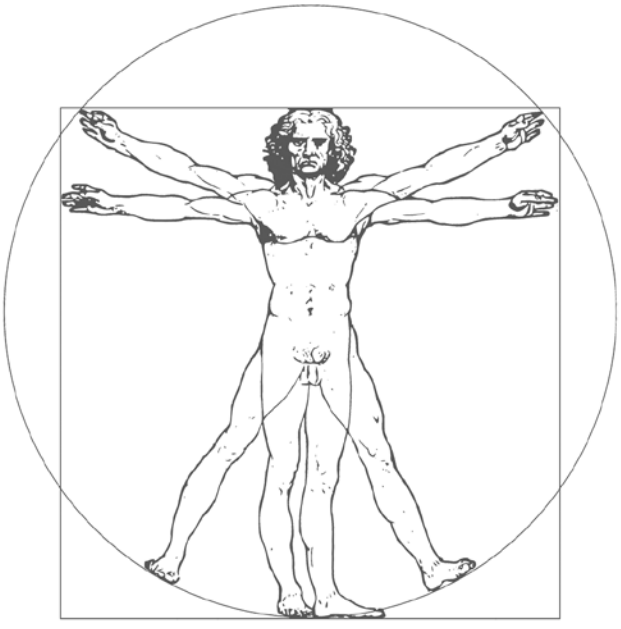

Abb. 1: Der ganzheitliche Mensch, wie Leonardo da Vinci ihn sah.

von der Softwareerstellung angewandt werden kann und der keine Aussage darüber trifft, mit welchen Mitteln Anforderungen in Software umgesetzt werden sollen.

Scrum ohne Unterbau

Die Erfahrung der letzten Jahre hat gezeigt, dass es vielen Scrum-Teams nicht gelingt, Sprint-Ergebnisse mit ausreichend hoher Qualität zu erstellen. Martin Fowler bezeichnete dies in [Fow09] als *Flaccid Scrum* und auch Ken Schwaber gestand in einem Interview ein, dass viele Entwicklungsteams nicht ausreichend mit modernen Entwicklungspraktiken vertraut sind (vgl. [Inf]).

Auch die Scrum Alliance hat diesen Missstand erkannt und eine neue Zertifizierung, den *Certified Scrum Developer (CSD)*, definiert. Zusätzlich zu der Kenntnis des Scrum-Prozesses muss ein CSD auch weitere technische Fähigkeiten zur Erstellung von Software erlernen.

Grundsätzlicher Nutzen und Schaden von Zertifizierungen wurden unter anderem in [Col09] diskutiert, ohne dass wir es hier wiederholen wollen. Ob die CSD-Zertifizierung als Werbeetikett sinnvoll ist, sollte jeder für sich selbst entscheiden. Unseren Erfahrungen nach ist die mit dem CSD-Zertifikat verbundene Schulung aber so wertvoll, dass wir sie jedem Entwickler – gerade im Scrum-Kontext – ans Herz legen.

Kasten 1: Braucht die Welt eine weitere Zertifizierung?

Praktiken eines Scrum-Entwicklers

Insbesondere Fähigkeiten und Kenntnisse in den im Folgenden beschriebenen Bereichen sind notwendig, um als CSD im Scrum-Kontext qualitativ hochwertige Software erstellen zu können.

Architektur und Design

Scrum erfordert flexible Strukturen, damit das Team jederzeit auf geänderte Anforderungen reagieren kann, die der *Product Owner (PO)* einbringt. Ohne flexible Strukturen kommt es zu der gefürchteten steilen Aufwandskurve, bei der Änderungen und komplett neue Features mit dem Fortschritt der Entwicklung so teuer werden, dass sie schließlich nicht mehr vertretbar umzusetzen sind. Ein CSD muss Entwurfsprinzipien kennen und anwenden können, die zu entkoppelten Strukturen führen.

Test Driven Development (TDD)

Die Grundregeln von TDD sind sehr einfach. Bob Martin hat sie in [Mar] auf nur drei Gesetze heruntergebrochen:

- Schreibe keinen Produktivcode ohne einen fehlschlagenden Test.
- Schreibe nur soviel Testcode, bis du einen fehlschlagenden Test hast.
- Du darfst nur genau so viel Produktivcode schreiben, wie gerade notwendig ist, um den einen fehlschlagenden Test zu erfüllen.

Jeder, der versucht hat, konsequent TDD zu betreiben, hat jedoch gemerkt, wie herausfordernd diese Regeln sind. So setzen viele Teams nur ansatzweise TDD um und kommen nicht dazu, sein eigentliches Potenzial auszuschöpfen. Die konsequente Anwendung von TDD-Praktiken führt zu flexiblem Design und erzeugt ein Sicherheitsnetz für Refaktorisierungen und funktionale Umstrukturierungen.

Continuous Integration

Um permanentes Feedback über den Entwicklungsstand zu bekommen, muss ein gemeinsames Quellcode-Repository vorhanden sein. Darauf basierend müssen automatisierte, getestete Builds mit hoher Geschwindigkeit erstellt werden können. Das ist der erste, fundamentale Schritt, um letztendlich zu *Continuous Deployment* (vgl. [Rie09]) zu gelangen, wie es von immer mehr Entwicklungsteams betrieben wird.

Collaboration

Scrum fordert und fördert eine enge Zusammenarbeit – sowohl innerhalb des Entwicklerteams, als auch mit dem PO und Stakeholdern. Wie dies geschehen kann, ist jedoch nicht näher festgehalten. So kommt es im oder kurz vor dem Sprint-Review immer wieder zu Überraschungen, was implizit angenommen und nicht klar kommuniziert wurde.

Ein CSD muss dagegen in der Lage sein, kommunikativ sowohl innerhalb des Teams (z. B. mit Pair-Programming) als auch mit dem Kunden (z. B. durch das Schreiben von automatisierten Akzeptanztests) zusammenzuarbeiten.

Refaktorisierung

So, wie der PO erst mit der Zeit eine Vorstellung davon gewinnt, welches die wertvollsten Features einer Software sind, bekommt auch das Entwicklungsteam erst mit der Zeit eine Vorstellung davon, welches die passende Architektur ist, um diese Features bestmöglich umzusetzen. Viele Entwickler schieben die Fortentwicklung der Architektur vor sich her – und das vor allem aus zwei Gründen:

- Es fällt ihnen schwer zu entscheiden, wann eine Refaktorisierung angebracht ist.
- Es fällt ihnen schwer, einen sicheren, zügigen Weg zu einer besseren Architektur zu finden, insbesondere falls dieser den begrenzten Rahmen der in integrierten Entwicklungsumgebungen eingebauten Refaktorisierungsmöglichkeiten sprengt.

Abb. 2: Das CSD-Zertifizierungsschema.

Ein CSD muss wissen, wann und wie Refaktorisierungen durchgeführt werden sollen, um die Architektur zu gestalten und fortzuentwickeln.

Für die Zertifizierung als CSD fordert die Scrum Alliance, dass ein Entwickler sowohl den Scrum-Prozess, als auch agile Entwicklungspraktiken kennt (vgl. [Scr]). Ein verbreiteter Weg zur CSD-Zertifizierung besteht darin, die Scrum-Grundlagen über einen CSM-Kurs und die Entwicklungspraktiken über einen *AEP*-Kurs (*Agile Entwicklungspraktiken*) zu lernen.

Alternativ zum CSM-Kurs kann ein Entwickler auch einen CSPO-Kurs (*Certified Scrum Product Owner*) oder eine Einführung in die *Scrum-Grundlagen (SG)* sowie eine zusätzliche *Wahlpflichtvertiefung (WV)* besuchen, wie in **Abbildung 2** zu sehen ist. Viele Entwickler haben jedoch schon einen CSM-Kurs besucht, um die Scrum-Grundlagen kennenzulernen, selbst wenn sie nicht als Scrum-Master arbeiten.

***Kasten 2:** Die CSD-Zertifizierung.*

Der ganzheitliche Entwickler

Ein CSD ist damit ein „ganzheitlicher" Entwickler. Er kennt den Scrum-Prozess und die XP-Entwicklungspraktiken und kann damit qualitativ hochwertige Sprint-Inkremente erstellen.

Damit wurde für Scrum nicht wirklich etwas Neues gefunden, sondern – wie schon XP – entdeckt der CSD bereits bekannte Praktiken neu. Warum? Weil wir merken, was für große Probleme wir haben, wenn wir diese Praktiken nicht verfolgen. Und weil wir wieder merken, dass dies kein Selbstläufer ist, sondern wir uns als Entwickler mit diesen Praktiken allein und im Team immer wieder auseinandersetzen müssen. ■

Literatur & Links

[Bec00] K. Beck, Extreme Programming Explained, Addison-Wesley 2000
[Col09] J. Coldewey, J. Link, Die Ent-Zertifizierung des Fortschritts, in: OBJEKTspektrum 03/2009
[Fow09] M. Fowler, Flaccid Scrum, 2009, siehe: http://martinfowler.com/bliki/FlaccidScrum.html
[Inf] InfoQ, Interview with Ken Schwaber – Part 3, siehe: http://www.infoq.com/news/2010/09/kenschwaber-interview-part3
[Mar] B. Martin, The Three Laws of TDD, siehe: http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd
[Rie09] E. Ries, Continuous deployment in 5 easy steps, 2009, siehe: http://radar.oreilly.com/2009/03/continuous-deployment-5-eas.html
[Sch01] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall 2001
[Scr] Scrum Alliance, Certified Scrum Developer, siehe: http://www.scrumalliance.org/CSD

Der Beitrag wurde in der Printausgabe 01/2012 des OBJEKTspektrum schon einmal veröffentlicht. Kurz vor Drucklegung hat sich die Scrum Alliance leider dazu entschlossen, den CSPO nicht mehr für eine Zertfizierung als CSD anzuerkennen.